PLOTTING SERIAL DATA FROM ARDUINO IN REAL TIME WITH PYTHON

Posted on February 4, 2018

Table of Contents

In this section, we will focus on sending data from the Arduino to the computer over a serial connection, and then plotting it with Python. A serial connection is basically a protocol that specifies how the messages are going to be sent. There's no need to know how the protocol works in order for us to accomplish what we are going to do here so let us just ignore it for now. For those of you who are interested, here's an **excellent article about serial communication**.

We will use the data from a potentiometer as an example for the code below since it involves only a simple `analogRead()`. In addition, I have limited the scope of this post to just sending ***float*** and ***int*** data types since these 2 data types will be sufficient for most applications. So without further ado, lets start off with the Arduino's code.

Arduino Code

For Arduino, an ***int*** data type is made up of 2 bytes, and a ***float*** data type is made up of 4 bytes. There's also the ***double*** data type that is usually 8 bytes long, but for the Arduino system, a ***double*** is exactly the same as a ***float*** (except for Arduino Due where it is 8 bytes) so it is also made up of 4 bytes. I wrote all my codes using ***double*** so take note that they are exactly the same as a ***float*** data type.

Now, for many applications, we want to collect data at a constant interval. This can be accomplished by creating a function that waits till a specified time before the function exits. Below is the full code for the Arduino. Take note of the `timeSync()` function which controls when the first line of code is executed.

```
unsigned long timer = 0;
long loopTime = 5000;   // microseconds

void setup() {
  Serial.begin(38400);
  timer = micros();
}

void loop() {
  timeSync(loopTime);
  //int val = analogRead(0) - 512;
  double val = (analogRead(0) -512) / 512.0;
  sendToPC(&val);
}

void timeSync(unsigned long deltaT)
{
  unsigned long currTime = micros();
  long timeToDelay = deltaT - (currTime - timer);
  if (timeToDelay > 5000)
  {
    delay(timeToDelay / 1000);
    delayMicroseconds(timeToDelay % 1000);
  }
  else if (timeToDelay > 0)
  {
    delayMicroseconds(timeToDelay);
  }
  else
  {
      // timeToDelay is negative so we start immediately
  }
  timer = currTime + timeToDelay;
}

void sendToPC(int* data)
{
  byte* byteData = (byte*)(data);
  Serial.write(byteData, 2);
}

void sendToPC(double* data)
{
  byte* byteData = (byte*)(data);
  Serial.write(byteData, 4);
}
```

The `timeSync()` function is very simple to implement. You can call it at the start of your loop such as that shown above, and it will do the job for you. In the `timeSync()` function, the value of 5000 is arbitrary. If you take a look at the **Arduino reference** for the `delayMicroseconds()` function, it says that the largest argument that will produce an accurate delay is 16383. So any value below this value should do.

Now that we have the time synchronization function, let us move on to the actual sending of the data. The sending of data through Arduino's serial function occurs in units of bytes. As such, we have to break up our ***int***/***double*** data into the bytes that it is composed of. This is actually simpler than you think if you

know how to use pointers.

A pointer is an address to a variable, and the type of the pointer defines how much to "jump" when the pointer is incremented. For example, an ***integer*** is 2 bytes long so when you increment an integer's pointer, it will "jump" 2 address values. On the other hand, a ***float*** is 4 bytes long so if you increment a float's pointer, it will increment by 4 address values as well. Now, what if we forcefully change an integer's pointer type to a byte type such as shown below?

```
int value = 234;
int* intPointer = &value;
byte* bytePointer = (byte*)(intPointer);
```

By doing this, we will be able to extract the byte composition from the original `value` variable. Since an ***int*** is made up of 2 bytes, `bytePointer[0]` will give the first byte, and `bytePointer[1]` will give the second byte. Below is the actual code snippet for converting ***int***/***double*** types into the ***bytes*** type.

```
void sendToPC(int* data)
{
  byte* byteData = (byte*)(data);    // Casting to a byte pointer
  Serial.write(byteData, 2);         // Send through Serial to the PC
}

void sendToPC(double* data)
{
  byte* byteData = (byte*)(data);    // Casting to a byte pointer
  Serial.write(byteData, 4);         // Send through Serial to the PC
}
```

Looks extremely simple isn't it? All we have to do is to take our data pointer, cast it to a byte pointer, then pass it and its original data size (in units of bytes) to Arduino's `Serial.write()` function. That's all for the Arduino portion. Let us now move on to the Python part of the code.

Python Code

Next up is thePython portion of the code. If you have not downloaded aPython IDE for your computer, I highly recommend Pycharm. You can check out the installation instructions from my **previous post**.

The Python portion is not as simple as the Arduino's one. We will create a custom class to handle the data from Arduino, and then use matplotlib to plot the graphs in real time. In order to send/receive Serial data, we will require another package called **pySerial**. This is not included within Anaconda so you have to download it manually. There are installation instructions for PySerial in their **website** but basically, you just have to type the following command in the terminal:

```
pip install pyserial
```

For Linux users, typing it in the terminal should do the trick but for Windows users, you probably have to use the Anaconda Prompt (typing anaconda in windows search should show it). Once you are done with this, we are good to go.

We are going to deal with the Serial data and plotting separately. In order to get the most updated values, data coming from the Arduino should be read as fast as possible. The buffer for the Serial data works in a FIFO (first in first out) manner so if we are not fast enough, we will be reading old values while data points accumulate in the buffer. To do this, we are going to handle the Serial data on a different thread. This thread will only have one job, and that is to continually read the Serial data and store it within a variable. The main thread will then read this variable, process it if needed, and plot it on a real time graph.

Below is the code for our Python real time graph plotter.

```
#!/usr/bin/env python

from threading import Thread
import serial
import time
import collections
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import struct
import pandas as pd


class serialPlot:
    def __init__(self, serialPort = '/dev/ttyUSB0', serialBaud = 38400, plotLength = 100
        self.port = serialPort
        self.baud = serialBaud
        self.plotMaxLength = plotLength
        self.dataNumBytes = dataNumBytes
        self.rawData = bytearray(dataNumBytes)
        self.data = collections.deque([0] * plotLength, maxlen=plotLength)
        self.isRun = True
        self.isReceiving = False
        self.thread = None
        self.plotTimer = 0
        self.previousTimer = 0
        # self.csvData = []

        print('Trying to connect to: ' + str(serialPort) + ' at ' + str(serialBaud) + '
        try:
            self.serialConnection = serial.Serial(serialPort, serialBaud, timeout=4)
            print('Connected to ' + str(serialPort) + ' at ' + str(serialBaud) + ' BAUD.
        except:
            print("Failed to connect with " + str(serialPort) + ' at ' + str(serialBaud)

    def readSerialStart(self):
        if self.thread == None:
            self.thread = Thread(target=self.backgroundThread)
            self.thread.start()
            # Block till we start receiving values
            while self.isReceiving != True:
                time.sleep(0.1)

    def getSerialData(self, frame, lines, lineValueText, lineLabel, timeText):
        currentTimer = time.perf_counter()
        self.plotTimer = int((currentTimer - self.previousTimer) * 1000)     # the first
        self.previousTimer = currentTimer
        timeText.set_text('Plot Interval = ' + str(self.plotTimer) + 'ms')
        value,  = struct.unpack('f', self.rawData)    # use 'h' for a 2 byte integer
        self.data.append(value)    # we get the latest data point and append it to our a
        lines.set_data(range(self.plotMaxLength), self.data)
        lineValueText.set_text('[' + lineLabel + '] = ' + str(value))
        # self.csvData.append(self.data[-1])

    def backgroundThread(self):    # retrieve data
        time.sleep(1.0)  # give some buffer time for retrieving data
        self.serialConnection.reset_input_buffer()
        while (self.isRun):
            self.serialConnection.readinto(self.rawData)
```

```
            self.isReceiving = True
            #print(self.rawData)

    def close(self):
        self.isRun = False
        self.thread.join()
        self.serialConnection.close()
        print('Disconnected...')
        # df = pd.DataFrame(self.csvData)
        # df.to_csv('/home/rikisenia/Desktop/data.csv')


def main():
    # portName = 'COM5'     # for windows users
    portName = '/dev/ttyUSB0'
    baudRate = 38400
    maxPlotLength = 100
    dataNumBytes = 4        # number of bytes of 1 data point
    s = serialPlot(portName, baudRate, maxPlotLength, dataNumBytes)   # initializes all
    s.readSerialStart()                                               # starts backgroun

    # plotting starts below
    pltInterval = 50    # Period at which the plot animation updates [ms]
    xmin = 0
    xmax = maxPlotLength
    ymin = -(1)
    ymax = 1
    fig = plt.figure()
    ax = plt.axes(xlim=(xmin, xmax), ylim=(float(ymin - (ymax - ymin) / 10), float(ymax
    ax.set_title('Arduino Analog Read')
    ax.set_xlabel("time")
    ax.set_ylabel("AnalogRead Value")

    lineLabel = 'Potentiometer Value'
    timeText = ax.text(0.50, 0.95, '', transform=ax.transAxes)
    lines = ax.plot([], [], label=lineLabel)[0]
    lineValueText = ax.text(0.50, 0.90, '', transform=ax.transAxes)
    anim = animation.FuncAnimation(fig, s.getSerialData, fargs=(lines, lineValueText, li

    plt.legend(loc="upper left")
    plt.show()

    s.close()


if __name__ == '__main__':
    main()
```

* Many thanks to James Richardson for notifying me that time.clock() is now deprecated and using time.perf_counter() instead will allow the program to correctly calculate the time interval.

There are many things going on within the above code snippet but I'll try my best to explain the general flow. When the `serialPlot` class is instantiated, it will try to connect to the specified serial port at the specified baudrate through the `serial.Serial()` function. Once connection is successful, we will get a

reference to the serial connection in `self.serialConnection` and we can then start our background thread to retrieve data.

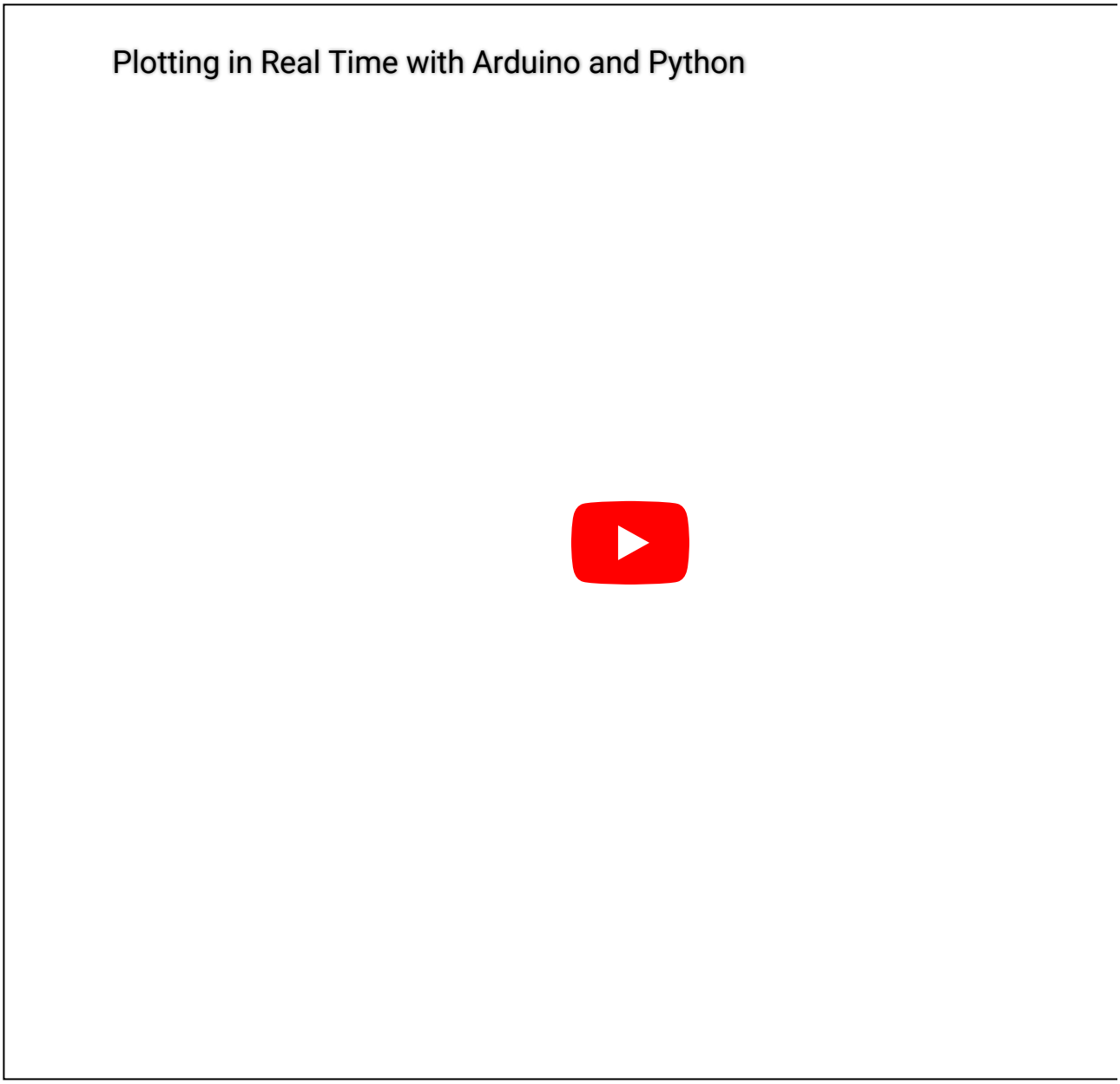
The background thread is started through calling the `readSerialStart(self)` function. Once it is started, it will continually read from the serial connection and assign the data to the self.rawData variable. In this manner, whenever the plotting function reads this variable, it will always have the latest data irregardless of the plotting rate (speed at which you plot the data).

Your computer screen should be functioning at 60Hz or 120Hz so even if you plot at a frequency higher than this, you will not be able to see it. As a result of this, you should always try to separate the data acquisition and plotting. There is no point in plotting faster than what your screen can display, and the plotting should not interfere (slow down) the data acquisition process as well.

Unlike C++, converting data type from ***bytes*** to ***double*** or ***integer*** in python is really simple using the struct library. As you can see in the code above, it was a one-liner. For other type of data conversion, you can take a look at this **reference page**. I have also included the capability to export the received data to a csv file in the code. Simply un-comment the commented code and it will save all the received data in the location that you specified when you exit the program. However, take note that exported csv is not time synchronized (each point is not a fixed duration away from the previous point).

Running the Programs

I have captured a video to show you a demo of the program. Hope this will be of help to you! (:



Running Multiple Plots at Once

Now that we are successful in plotting a single graph in real time with Python, let us move one step further to make multiple plots. Actually, this is extremely easy in Python. Most of the time, if you are successful with one, creating multiples of it simply involve putting the individual items in a list and then iterate through them. Let us take a look at the changes in the Arduino code. I am going to give an example of drawing 3 plots but it is actually possible to do any number that you desire with just some simple changes to the code.

```
void sendToPC(int* data1, int* data2, int* data3)
{
  byte* byteData1 = (byte*)(data1);
  byte* byteData2 = (byte*)(data2);
  byte* byteData3 = (byte*)(data3);
  byte buf[6] = {byteData1[0], byteData1[1],
                 byteData2[0], byteData2[1],
                 byteData3[0], byteData3[1]};
  Serial.write(buf, 6);
}

void sendToPC(double* data1, double* data2, double* data3)
{
  byte* byteData1 = (byte*)(data1);
  byte* byteData2 = (byte*)(data2);
  byte* byteData3 = (byte*)(data3);
  byte buf[12] = {byteData1[0], byteData1[1], byteData1[2], byteData1[3],
                  byteData2[0], byteData2[1], byteData2[2], byteData2[3],
                  byteData3[0], byteData3[1], byteData3[2], byteData3[3]};
  Serial.write(buf, 12);
}
```

As you can see from the code above, it isn't too much different from the original. We simply have to take all our byte data, put them together into a buffer and send it though Arduino's `Serial.write()` function as usual.

For Python's plotting, however, there are 2 different ways of doing things. You can have many plots in a single graph, or you can choose to have one plot for each individual graph. The choice is up to you and I will go through them both here.

Multiple Plots in a Single Graph

For the Python portion, we have to make multiple arrays to store the data from Arduino. This is actually simpler than in sounds. You can literally put anything into Python's "list" so let us exploit this "list". I believe that the changes in the code are quite self explanatory should I'll just shut up and post the code below 🙂 If you have any questions, feel free to write them in the comments section below and I'll get back to you as soon as possible.

```
#!/usr/bin/env python

from threading import Thread
import serial
import time
import collections
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import struct
import copy
import pandas as pd


class serialPlot:
    def __init__(self, serialPort='/dev/ttyUSB0', serialBaud=38400, plotLength=100, data
        self.port = serialPort
        self.baud = serialBaud
        self.plotMaxLength = plotLength
        self.dataNumBytes = dataNumBytes
        self.numPlots = numPlots
        self.rawData = bytearray(numPlots * dataNumBytes)
        self.dataType = None
        if dataNumBytes == 2:
            self.dataType = 'h'     # 2 byte integer
        elif dataNumBytes == 4:
            self.dataType = 'f'     # 4 byte float
        self.data = []
        for i in range(numPlots):   # give an array for each type of data and store them
            self.data.append(collections.deque([0] * plotLength, maxlen=plotLength))
        self.isRun = True
        self.isReceiving = False
        self.thread = None
        self.plotTimer = 0
        self.previousTimer = 0
        # self.csvData = []

        print('Trying to connect to: ' + str(serialPort) + ' at ' + str(serialBaud) + '
        try:
            self.serialConnection = serial.Serial(serialPort, serialBaud, timeout=4)
            print('Connected to ' + str(serialPort) + ' at ' + str(serialBaud) + ' BAUD.
        except:
            print("Failed to connect with " + str(serialPort) + ' at ' + str(serialBaud)

    def readSerialStart(self):
        if self.thread == None:
            self.thread = Thread(target=self.backgroundThread)
            self.thread.start()
            # Block till we start receiving values
            while self.isReceiving != True:
                time.sleep(0.1)

    def getSerialData(self, frame, lines, lineValueText, lineLabel, timeText):
        currentTimer = time.perf_counter()
        self.plotTimer = int((currentTimer - self.previousTimer) * 1000)     # the first
        self.previousTimer = currentTimer
        timeText.set_text('Plot Interval = ' + str(self.plotTimer) + 'ms')
        privateData = copy.deepcopy(self.rawData[:])    # so that the 3 values in our pl
        for i in range(self.numPlots):
```

```
            data = privateData[(i*self.dataNumBytes):(self.dataNumBytes + i*self.dataNum
            value,  = struct.unpack(self.dataType, data)
            self.data[i].append(value)    # we get the latest data point and append it t
            lines[i].set_data(range(self.plotMaxLength), self.data[i])
            lineValueText[i].set_text('[' + lineLabel[i] + '] = ' + str(value))
        # self.csvData.append([self.data[0][-1], self.data[1][-1], self.data[2][-1]])

    def backgroundThread(self):    # retrieve data
        time.sleep(1.0)  # give some buffer time for retrieving data
        self.serialConnection.reset_input_buffer()
        while (self.isRun):
            self.serialConnection.readinto(self.rawData)
            self.isReceiving = True
            #print(self.rawData)

    def close(self):
        self.isRun = False
        self.thread.join()
        self.serialConnection.close()
        print('Disconnected...')
        # df = pd.DataFrame(self.csvData)
        # df.to_csv('/home/rikisenia/Desktop/data.csv')


def main():
    # portName = 'COM5'
    portName = '/dev/ttyUSB0'
    baudRate = 38400
    maxPlotLength = 100     # number of points in x-axis of real time plot
    dataNumBytes = 4        # number of bytes of 1 data point
    numPlots = 3            # number of plots in 1 graph
    s = serialPlot(portName, baudRate, maxPlotLength, dataNumBytes, numPlots)   # initia
    s.readSerialStart()                                                    # starts backgroun

    # plotting starts below
    pltInterval = 50    # Period at which the plot animation updates [ms]
    xmin = 0
    xmax = maxPlotLength
    ymin = -(1)
    ymax = 1
    fig = plt.figure(figsize=(10, 8))
    ax = plt.axes(xlim=(xmin, xmax), ylim=(float(ymin - (ymax - ymin) / 10), float(ymax
    ax.set_title('Arduino Accelerometer')
    ax.set_xlabel("Time")
    ax.set_ylabel("Accelerometer Output")

    lineLabel = ['X', 'Y', 'Z']
    style = ['r-', 'c-', 'b-']  # linestyles for the different plots
    timeText = ax.text(0.70, 0.95, '', transform=ax.transAxes)
    lines = []
    lineValueText = []
    for i in range(numPlots):
        lines.append(ax.plot([], [], style[i], label=lineLabel[i])[0])
        lineValueText.append(ax.text(0.70, 0.90-i*0.05, '', transform=ax.transAxes))
    anim = animation.FuncAnimation(fig, s.getSerialData, fargs=(lines, lineValueText, li

    plt.legend(loc="upper left")
    plt.show()
```

```
    s.close()


if __name__ == '__main__':
    main()
```

Here's an example of how the plots look like when I used it to read values from an accelerator. The 3 lines corresponds to the x, y and z accelerations respectively.

Multiple Plots Using Multiple Graphs

In order to plot with multiple windows, you will need to keep multiple references to the animation object so that the window stays open. Instead of using a single `animation.FuncAnimation()` function like the cases before, we will need to call this function multiple times and store a reference of it to establish multiple plots. This is rather similar to the first example for a single plot except that we now need to call the `animation.FuncAnimation` multiple times (and we have to keep its reference somewhere too).

Take a look below for the full Python code. Once again, I believe it is quite self-explanatory so I'll leave it to you to read through the code. The main changes are in the `main()` function and the `getSerialData()` function.

```
#!/usr/bin/env python

from threading import Thread
import serial
import time
import collections
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import struct
import copy


class serialPlot:
    def __init__(self, serialPort='/dev/ttyUSB0', serialBaud=38400, plotLength=100, data
        self.port = serialPort
        self.baud = serialBaud
        self.plotMaxLength = plotLength
        self.dataNumBytes = dataNumBytes
        self.numPlots = numPlots
        self.rawData = bytearray(numPlots * dataNumBytes)
        self.dataType = None
        if dataNumBytes == 2:
            self.dataType = 'h'     # 2 byte integer
        elif dataNumBytes == 4:
            self.dataType = 'f'     # 4 byte float
        self.data = []
        self.privateData = None     # for storing a copy of the data so all plots are sy
        for i in range(numPlots):   # give an array for each type of data and store them
            self.data.append(collections.deque([0] * plotLength, maxlen=plotLength))
        self.isRun = True
        self.isReceiving = False
        self.thread = None
        self.plotTimer = 0
        self.previousTimer = 0

        print('Trying to connect to: ' + str(serialPort) + ' at ' + str(serialBaud) + '
        try:
            self.serialConnection = serial.Serial(serialPort, serialBaud, timeout=4)
            print('Connected to ' + str(serialPort) + ' at ' + str(serialBaud) + ' BAUD.
        except:
            print("Failed to connect with " + str(serialPort) + ' at ' + str(serialBaud)

    def readSerialStart(self):
        if self.thread == None:
            self.thread = Thread(target=self.backgroundThread)
            self.thread.start()
            # Block till we start receiving values
            while self.isReceiving != True:
                time.sleep(0.1)

    def getSerialData(self, frame, lines, lineValueText, lineLabel, timeText, pltNumber)
        if pltNumber == 0:  # in order to make all the clocks show the same reading
            currentTimer = time.perf_counter()
            self.plotTimer = int((currentTimer - self.previousTimer) * 1000)     # the f
            self.previousTimer = currentTimer
        self.privateData = copy.deepcopy(self.rawData)    # so that the 3 values in our
        timeText.set_text('Plot Interval = ' + str(self.plotTimer) + 'ms')
        data = self.privateData[(pltNumber*self.dataNumBytes):(self.dataNumBytes + pltNu
```

```
            value,  = struct.unpack(self.dataType, data)
            self.data[pltNumber].append(value)    # we get the latest data point and append
            lines.set_data(range(self.plotMaxLength), self.data[pltNumber])
            lineValueText.set_text('[' + lineLabel + '] = ' + str(value))

        def backgroundThread(self):    # retrieve data
            time.sleep(1.0)  # give some buffer time for retrieving data
            self.serialConnection.reset_input_buffer()
            while (self.isRun):
                self.serialConnection.readinto(self.rawData)
                self.isReceiving = True

        def close(self):
            self.isRun = False
            self.thread.join()
            self.serialConnection.close()
            print('Disconnected...')


    def makeFigure(xLimit, yLimit, title):
        xmin, xmax = xLimit
        ymin, ymax = yLimit
        fig = plt.figure()
        ax = plt.axes(xlim=(xmin, xmax), ylim=(int(ymin - (ymax - ymin) / 10), int(ymax + (y
        ax.set_title(title)
        ax.set_xlabel("Time")
        ax.set_ylabel("Accelerometer Output")
        return fig, ax


    def main():
        # portName = 'COM5'
        portName = '/dev/ttyUSB0'
        baudRate = 38400
        maxPlotLength = 100     # number of points in x-axis of real time plot
        dataNumBytes = 4        # number of bytes of 1 data point
        numPlots = 3            # number of plots in 1 graph
        s = serialPlot(portName, baudRate, maxPlotLength, dataNumBytes, numPlots)   # initia
        s.readSerialStart()                                                    # starts backgroun

        # plotting starts below
        pltInterval = 50    # Period at which the plot animation updates [ms]
        lineLabelText = ['X', 'Y', 'Z']
        title = ['X Acceleration', 'Y Acceleration', 'Z Acceleration']
        xLimit = [(0, maxPlotLength), (0, maxPlotLength), (0, maxPlotLength)]
        yLimit = [(-1, 1), (-1, 1), (-1, 1)]
        style = ['r-', 'g-', 'b-']    # linestyles for the different plots
        anim = []
        for i in range(numPlots):
            fig, ax = makeFigure(xLimit[i], yLimit[i], title[i])
            lines = ax.plot([], [], style[i], label=lineLabelText[i])[0]
            timeText = ax.text(0.50, 0.95, '', transform=ax.transAxes)
            lineValueText = ax.text(0.50, 0.90, '', transform=ax.transAxes)
            anim.append(animation.FuncAnimation(fig, s.getSerialData, fargs=(lines, lineValu
            plt.legend(loc="upper left")
        plt.show()

        s.close()
```

```
if __name__ == '__main__':
    main()
```

And here's a video of how the application looks like when you run it.

Go to Next Section

PREVIOUS POST
Building the Circuit and the Hardware

NEXT POST
Creating a Graphic User Interface (GUI) with Python

POSTED IN FEEDBACK CONTROL SYSTEM

47 comments on "*Plotting Serial Data from Arduino in Real Time with Python*"

Nitin Kisan Palte

March 12, 2018 at 12:34 am

great work

but when i run the code on raspberry pi it shows me single graph at a time.
when 1st one was closed 2nd graph was appeared and so on.

how to display all plot at same time as shown in video.

thanks

rikisenia.L

March 12, 2018 at 8:15 pm

Hi Nitin Kisan Palte

Thanks for reading my post.
The code that I have written was not tested on the raspberry pi so I cannot be sure what went wrong.

From your question, I assume that you are talking about the Multiple Plots using Multiple Graphs.
It seems to me like the indentation of the "plt.show()" command might be wrong.
Can you check if it is the same as what was written in the post?
"plt.show()" should be outside of the for loop so that the program will show all plots at once.

Alvaro Callejas

May 18, 2018 at 6:02 am

First of all thanks for sharing with such detail in the explanation.

I have an observation of the performance. When I run the first example (single graph, single plot) my CPU consumption goes nuts, always above 50%. Does this happened to you too? Have you monitored the performance?

I'm working on Ubuntu 18. Here's a proof screenshot, please replace the dots:
photos[dot]app[dot]goo[dot]gl/STv05tufK90V74rz2

rikisenia.L

May 18, 2018 at 8:18 am

Hi Alvaro Callejas

I have not checked it on my Ubuntu 16.04 system yet but on my windows system, it is running at below 5% CPU even when I am plotting 6 graphs in a single window. That being said, I am running my process on Ryzen 1700X which has 8 cores so it may not be a good comparison. In addition, the data is only sent at a frequency of 100Hz (100 times per second).

The background thread is written in such a way that it will run whenever there is data in the serial buffer (on the computer side). I think the problem may be because you are sending data to your computer too fast. Try putting a delay in between each serial write from your microprocessor and see if it helps to decrease the CPU usage.

On a side note, I believe that you are using [htop] in the screenshot. [htop] gives a max CPU utilization of number of cores * 100% so it is possible to get 400% CPU usage on system. Therefore, the total CPU utilization in your case may be actually 15%. (It is possible to show a max of 100% regardless of the number of cores using [top] but I am not sure if it can be done using [htop]).
Here's a link for a similar case: https://superuser.com/questions/457624/why-is-the-top-command-showing-a-cpu-usage-of-799
You can also check the man page for [top] for more details on the CPU usage.

tomy world

October 13, 2018 at 4:24 pm

The best examples of multiple data, real time plotting example found so far.
Thank you so much 😉

And I've been trying to figure how to have the x-axis update in real time, instead of the fixed scale (0-100).

Any hints?

rikisenia.L

October 13, 2018 at 5:15 pm

Hi Tomy,

Glad that my post helped you in some way.

The x-axis can be updated by passing the plot axes (ax) to the function responsible for updating the plot array (getSerialData), and then using the method ax.set_xlim() to set a new limit every time the plot updates. You will also have to update the data array to take in the "correct" x-values as well so that the plot will appear at the correct place.

[Example]
def getSerialData(self, frame, lines, lineValueText, lineLabel, timeText, pltInterval, ax):
...
self.counter += 1
lines.set_data(range(self.counter, self.counter + self.plotMaxLength), self.data)
ax.set_xlim(self.counter, self.plotMaxLength + self.counter)
...

However, do take note that the plotting interval is the one specified in the python code (pltInterval) and it may not match the interval at which the data is arriving. The program simply plots the latest data point received so be careful when interpreting the x-axis.

James Richardson

January 18, 2019 at 7:13 pm

Thanks for the great walk-through.

One thing I can't figure out, how come the plot interval printed in the figure isn't limited to be the same as the plot interval for the animation update? Surely they should be the same since that is the interval for the call to getserialdata()?

Any thoughts?

rikisenia.L

January 18, 2019 at 10:36 pm

Great insight! Actually I found it weird too so in the new version of the code which I did not upload, I deleted away the section for calculation the interval which is printed in the plot.
This behavior is probably because the starting and ending time of the function is not guaranteed, but the refresh time of the plot is guaranteed. In other words, getserialdata() function can start and return anytime, but will wait for the timer to reach a certain value before the plot will be refreshed. As such, any measurements made within the getserialdata() function will not be accurate.

** EDIT **
Sorry, I was wrong about this. After looking around more with the references given by James, I came upon the PEP418 which explains why time.clock() was deprecated.
Below is the abstract:
To measure the performance of a function, time.clock() can be used but it is very different on Windows and on Unix. On Windows, time.clock() includes time elapsed during sleep, whereas it does not on Unix. time.clock() resolution is very good on Windows, but very bad on Unix. The new time.perf_counter() function should be used instead to always get the most precise performance counter with a portable behaviour (ex: include time spend during sleep).

On linux, time.clock() does not measure the time spent during sleep thus the values shown on the graph is actually the time spent processing, and not the time interval. Once I switched to time.perf_counter(), values that are really close to the time interval set were observed (I set 50ms interval and the measured time was approximately 49~50ms). time.process_time() excludes the sleep time too so it gives the approximately the same value as time.clock().

I tried the above on windows as well but was unable to get good results. All 3 methods gave me ~60ms even when I set the time interval as 50ms. Let me know if anyone has something on this.

Serdar Sen

January 31, 2019 at 7:38 am

Hey,

thanks a lot for the great guide for plotting serial data. If i want to do multiple plots in one window, everythings is fine. I've adapted your Arduino and Python code. But when i try to Plot in multiple windows i just get one sensor value in one window and the other ones just show nothing. Is there a possibility that your last code is wrong? Actually i want to have 3 Plot windows, the first with 3 sensor values and the other two with a single value. I adapted your Arduino Code to send 5 values and plot them in one Window. I hope you can help me. Thanks a lot for your support and great work.

rikisenia.L

January 31, 2019 at 9:46 pm

Hi Serdar Sen,

You're right. There has been an indentation error on line 56 which probably caused the error that you are seeing. I guess I carelessly added in a tab while porting the code. Well, this also meant that you're the first person who really tried out the code! Thanks!
Anyway, I have corrected and tested the code in the post so it should work now.
Also, I changed the function that gets the time since time.clock() is now deprecated.

Hope this helps!

Serdar Sen

February 1, 2019 at 6:00 pm

Hey,

thanks for the quick response! Now i can see 5 values in 5 Windows. I've tried a lot to adapt your code to plot 4 values in the first window but it won't work. My Arduino is already sending 5 float values out. Do you have any ideas how to realize this?

rikisenia.L

February 1, 2019 at 10:22 pm

Hi Serdar,

I'm glad it worked!
What you are trying achieve will be a little complicated to do.
If all the windows are plotting the same number of variables (for example 2 windows which plot 3 lines each), the structure of the code can be the same and this will be relatively simple to do.
However, plotting 4 variables in 1 window and 1 variable in the second window will require quite a huge change to the code.

gustavo
March 14, 2019 at 2:31 pm

Hi, i want graph three waves in one graph, and others trhee waves in other graph, thanks for the help.

rikisenia.L
March 14, 2019 at 10:14 pm

Hi gustavo,

Perhaps you can tell me what are the problems that you have encountered while modifying my code so that I know how to help you.

Gusito
March 15, 2019 at 4:07 am

Thank you!
I'm new in Python, its code Works very well, Im tryng tom make an application where I need to make two graphics, and each of them has 3 plots, like the second example, but with two equals graphics, and I have not managed to do it, would appreciate very much you help, gretting

Jeff M.

April 8, 2019 at 1:11 pm

Hello,

Very impressed by your website. I am just finishing up my undergrad in Aerospace Engineering, but want to continue school/professional work in mechatronic systems.

I had a quick question about the time interval of plotting this data. My current “fun side project” is I want to develop a drone from scratch and I am running into the issue where I need to see the data running in the microcontroller real-time, thus me finding your website. This data is running at 250 Hz. How would you recommend using your code for my application? (I ask because I see the delta t on the matplotlib plots) or do you believe it should work fine? Could I possibly only send every third (or forth, fifth, etc.) packet of data from the microcontroller so the Python script runs and it doesn’t bottleneck the system? I also will inevitably need to save this data to a file as well, but Ill get there when I get there 🙂

Thank you and have a great rest of your day!

rikisenia.L

April 8, 2019 at 10:43 pm

Hi Jeff,

Thanks for the compliment.

Unfortunately, it is not really possible to visualize a 250Hz data because our eyes cannot track such fast changes (fluorescent lights flicker at around 120Hz but we cannot detect the flicker) and the average monitor refreshes at 60Hz.

The python script that I have written reads incoming data as fast as it can and plots the newest data everytime the display updates (50ms interval = 20Hz). I have not tested the actual upper limit of the serial transfer but I think 250Hz should work just fine assuming you are transferring a 4 byte variable with a baudrate of 38400 (theoretically you should be able to transfer at 38400/(4*8) = 1200Hz but in practice it will be lower). In the python script above, there is already a function provided to store and export the data to a csv file format. Perhaps you can try with a simple sine function from the microcontroller at 250Hz, then export the collected data to csv and check if any values are missing to confirm that a transfer at 250Hz is possible.

Hope this helps! Cheers!

Oliver A.
June 5, 2019 at 1:26 am

Hi,

I attempted running your code and animation seems to run once before disconnecting from the serial. Any idea why this happens?

Thanks!

rikisenia.L
June 5, 2019 at 8:56 pm

Hi Oliver,

What do you mean by the animation running once before disconnecting?
Can you please elaborate more about the problem?

Oliver A.
June 6, 2019 at 2:14 am

Hi,

So I don't think the animation ever actually starts running. The plot pops up with no data. Doing some further debugging, it appears that getSerialData function never actually runs. The backgroundThread runs for a short time then disconnects.

Thanks

Oliver A.
June 6, 2019 at 6:06 am

Hi,

So after spending some time debugging, I was able to make the code run by removing the main() function and simply running the code inside the if __name__ == '__main__': conditional. Do you have any idea why this is the case?

Thanks

rikisenia.L
June 6, 2019 at 8:51 pm

Hi Oliver,

I am not sure what the problem is as well.
Since the code runs if it is placed under the if __name__ == '__main__': condition, main() should be called as well if it is placed under it.
Can you try running the file with minimal changes to it again?

Ammar chauhan

August 24, 2019 at 2:30 am

Hi
Nice Post , can you please share the Mpu6050 code as well for arduino ide ??
Thanks
Cheers
Ammar

rikisenia.L

August 24, 2019 at 10:22 am

Hi Ammar,

I was not using the MPU6050 for the sensor but LSM303D and L3GD20H.
If you still want the code for the sensors that I was using, it is available in section 8 of the post (https://thepoorengineer.com/en/angle-control-absolute/)

Puja chaudhari

October 3, 2019 at 7:36 pm

Hi,
I want to read all analogue values(7) in a string from the Arduino and that I want to plot in the python.
how will be the code for Arduino and python will change. Can you send me code regarding this for both?

rikisenia.L

October 4, 2019 at 9:26 pm

Hi,

If you go through the tutorial thoroughly, I am sure you will know where to change the code to do what you want to do. I will not write the code for you, but if you run into any problems while trying, be specific about which part and I'll definitely help you out 🙂

Rudy Coppens

November 28, 2019 at 5:48 am

Hi,

FIrst of all thx for your code! I want to use it to make a data plotter for a project.
I tried it now for 3 seperated graphs and it works. However at 3 plots my laptop with i7-4600U
(not the fastest but also not the slowest) is running into CPU bottleneck.
I tried already some things such as decreasing the Baud and adding more delay in the Arduino code but it looks like python is the problem. Do you have maybe anty idea what I can change?

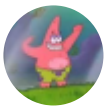
Thx.

rikisenia.L

March 17, 2020 at 4:26 pm

Hi Rudy,

Apologies for the late reply.
Actually I would recommend decreasing the number of graphs and just plot them in the same graph if you can.
Also, if you don't require it to be in real time, you can simply save the data points and plot from the csv file later on.



Cesar Torres

December 31, 2019 at 3:57 pm

Hi,

First of all… Great Work !!!! I needed a tutorial with threads in python… thank youuuuu !.

I'm working with an arduino DUE, I changed 4 to 8 (as your mentioned in this tutorial for arduino DUE) bytes but my graph in the first code is not good… Lines appear in my graph and potentiometer value ranges between -2 and 13 almost instantly (maybe for this reason appear lines in my graph) for one end of the potentiometer and 0 – 1.8 for the other…
I would love read your answer… Happy New Year 🙂

rikisenia.L

March 17, 2020 at 4:21 pm

Hi Cesar,

Apologies for the late reply.
You will have to modify the Arduino sendToPC() function as well to send 8 bytes of data instead of 4 bytes for each data point.
I hope this reply is still in time to help you..

Insanoff

February 4, 2020 at 11:03 pm

I have tried with Arduino Pro Mini, leaving dataNumBytes equal to 4. Values are jumping 0 and -4.25366E+52. I am trying to figure out.
Excellent write-up!

Insanoff

February 5, 2020 at 1:06 am

After struggling an hour, I restarted my laptop and got everything work. 😀 Very strange.

Insanoff

February 5, 2020 at 1:38 am

One problem I have noticed, sometimes it does not work properly and shows gibberish. Probably because it does not know where bit starts.

rikisenia.L

March 17, 2020 at 4:18 pm

Hi Insanoff

Apologies for the late reply.
You are absolutely right on this. I have seen this behaviour multiple times myself as well.
I noticed that it happens more frequently when your baudrate setting is high.
I don’t really have a solution to this but restarting the program several times should solve the issue.

Andrew

March 6, 2020 at 9:27 pm

What a brilliant walk through! thanks so much

I have an Arduino that continuously spits out values, ranging from 0-90 and I have altered the python code to get it to display these values on a live plot. The problem is that I am only able to get this to work when I use the int data type i.e. I cannot get it to work with the double data type.

When I try to live plot the same values as doubles, I get crazy values that rapidly go between positive and negative. e.g. +2.343535665e30, +3.34564565435e15, -1.98646334432e18

Yet when I print my values to the Arduinos serial monitor, they appear to be normal.

Have you any ideas what could be causing this change?

Thanks and kind regards,
Andrew

rikisenia.L

March 17, 2020 at 4:16 pm

Hi Andrew,

Apologies for the late reply. I have been abroad for quite some time due to my work.
Did you make sure to change the settings on the python (dataNumBytes=4 for double) side as well?
Sometimes, the crazy values are due to the wrong synchronization between python and the serial interface.
Try to stop and start the code a few times to see if it solves the problem.

Luis

March 18, 2020 at 10:16 am

Hello!
I am performing data acquisition at 1000Hz or I can store 1000 samples in 1 second in the CSV.
can you tell me if it is possible to sample these data in the graph in 1 second ??

rikisenia.L

April 7, 2020 at 9:18 pm

Hi Luis,

Apologies for the late reply. Assuming you are transferring a 4 byte variable with a baudrate of 38400, theoretically you should be able to transfer at 38400/(4*8) = 1200Hz but in practice it will be lower.
If you are just using 2 byte integer, sampling at 1000Hz and storing the data should not be a problem. However, it will be meaningless to plot all the 1000 points in a second because your screen can probably only refresh at a rate of 60Hz, so even if the program can do it, you will not be able to visualize it.

Eisa

April 7, 2020 at 3:28 pm

First of all, thank you so much for the neat work.

I only have a question, and it might be stupid, but I just started.

So, if I write the code in Python, do I need to run the serial on the ARDUINO IDE ? or from this method, there is no need to run the serial from the Arduino IDE ?

Thank you so much in advance,,

rikisenia.L

April 7, 2020 at 9:20 pm

Hi Eisa,

No, you do not have to operate the Aruino IDE at all after you finished transferring the code to the Arduino.
Hope my article helped you in some way!

Genoz

May 4, 2020 at 12:28 pm

hi, can you help me ?

how can i solved this problem?

File “led.py”, line 177, in
main()
File “led.py”, line 158, in main
app = Window(fig, root, s)
File “led.py”, line 94, in __init__
self.initWindow(figure) # initialize the window with our settings
File “led.py”, line 98, in initWindow
canvas = FigureCanvasQTAgg(figure, master=self.master)
TypeError: __init__() got an unexpected keyword argument ‘master’

frank

September 4, 2020 at 8:32 pm

Hi thank you so much for your great work.

I am trying to create a serial tool which includes a real-time plotting feature.
my arduino demo code is simply using Serial.println(“a”).(a is a float number)
but i have few issues with plotting.
is that possible that i could send you the code and get some tips from you.

Santi

October 25, 2020 at 8:45 am

Hey thanks for sharing.

Im having trouble to sample the data of a 12 bit resolution ADC (im using a LPC1769 board) does anyone know how I should change de code to work with this?

Should I change the thread Im sending from the device (currently sendig 2 packages of 6 bits each (the 6 least significant bits of the conversion and the 6 most significant ones)?

Why do you convert to double the conversion of de ADC (analogRead(0) -512) / 512.0;) ?

Nicyeung

December 6, 2020 at 8:17 pm

Hi, I want to use this to dispaly temperature from Serial('COM1', 9600), because it is my temperature reader port. But after i change the serial, the y_axis displays a line and when it doesn't change when the temperature. I'm confused where is the problem. My temperature sensor is from Omron. Thanks!

Nicyeung

December 6, 2020 at 8:19 pm

Sorry, the question is the curve doesn't change when the temperature changes.

H.W.

December 21, 2020 at 4:08 am

Thank your for your really nice post. Saved me hours or day figuring out how to do it on my own.

Trying out your code ploting simple sin-cuves from the arduino I got to the point, where sometimes I run into synchronisation-problems of the transferred data, as there is no sync-mechanism.

So I want to change the code to exchanges the values printing the values as text, separating the values by tabs and snchronizing on the new-line. Something like:
0.25\t-0.56\t0.99\n

I know the I need to transfer more bytes, but my data-rate is really slow compared to 115200 baud-rate I'm using. So efficiency for datatransfer is no issue for me.

The changes to the arduino-code is very simple, and I can handle that. But I'm quite new to python and I'm struggling with the code in the function "getSerialData".

Could you give me some advice or even a code-example.

Thanks.

ruben

May 25, 2021 at 7:45 pm

Brilliant.
As a newbie on this, I was stuck for a long while why the animation didnt work, but eventually I found out that it was due to the IDE I was using (Spider, jupyter), I changed and all worked well. Many thanks!

Comments are closed.

Categories

Select Category

Archives

- September 2019 (3)
- February 2019 (1)
- January 2019 (1)
- September 2018 (1)
- July 2018 (4)
- March 2018 (3)
- February 2018 (5)
- November 2017 (6)
- July 2017 (2)
- April 2017 (6)
- March 2017 (4)
- February 2017 (2)
- January 2017 (9)
- December 2016 (24)
- November 2016 (1)